02

Autenticação através de Shiro

Download

Caso queira começar o treinamento a partir desse vídeo, pode baixar o projeto aqui (https://s3.amazonaws.com/caelum-online-public/auron/auron-stage8.zip). Só baixe este arquivo se não tiver feito os exercícios dos capítulos anteriores.

Página de login

Agora que temos o cadastro de participantes e o algoritmo de sorteio, vamos implementar a *autenticação* no sistema. Somente usuários logados poderão criar novos sorteios.

Vamos começar a implementar a tela de *login*. Aqui não há novidade e como já criamos duas telas XHTML durante o treinamento colamos o código pronto para este video.

```
<?xml version="1.0" encoding="US-ASCII" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:p="http://primefaces.org/ui">
<h:head>
<meta http-equiv="Content-Type" content="text/html; charset=US-ASCII" />
<title>Insert title here</title>
</h:head>
<h:body>
    <h:form>
        <p:panelGrid columns="2">
            <p:outputLabel value="Email: "/>
            <p:inputText value="#{participanteBean.participante.email}" />

            <p:outputLabel value="Senha: " />
            <p:password value="#{participanteBean.participante.senha}" />

            <p:commandButton value="Logar" action="#{participanteBean.login}" ajax="false" />

        </p:panelGrid>
    </h:form>
</h:body>
</html>
```

Vamos criar um novo método no `ParticipanteBean` chamado `login()`. Por enquanto, este método imprimirá na tela somente o *email* do participante que foi preenchido no formulário.

```
public void login() {
      System.out.println(participante.getEmail());
}
```

Autenticação com Apache Shiro

Começaremos agora a implementar autenticação com o *Apache Shiro*. Para utilizá-lo, vamos adicionar suas dependências no arquivo `pom.xml` .

```
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-core</artifactId>
    <version>1.2.1</version>
</dependency>
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-web</artifactId>
    <version>1.2.1</version>
</dependency>
```

Após isso, precisamos adicionar o *filtro* e o *listener* do Apache Shiro no arquivo `web.xml` . O Apache Shiro é nada mais do que um Filtro do especificação Servlet que recebe todas as requisições para verificar as permissões do usuário:

```
<listener>
    <listener-class>org.apache.shiro.web.env.EnvironmentLoaderListener</listener-class>
</listener>

<filter>
    <filter-name>ShiroFilter</filter-name>
    <filter-class>org.apache.shiro.web.servlet.ShiroFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>ShiroFilter</filter-name>
    <url-pattern>/*</url-pattern>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>INCLUDE</dispatcher>
    <dispatcher>ERROR</dispatcher>
</filter-mapping>
```

O arquivo shiro.ini

Precisamos também criar um arquivo onde faremos as configurações básicas do *Shiro*. Tais como: definir quem serão os usuários, a *URL* que queremos interceptar e a página de *login*. Este arquivo, deverá se chamar `shiro.ini` e deve estar localizado na pasta `WEB-INF` .

O arquivo `shiro.ini` é dividido em *seções*. Vamos começar configurando a seção **main**. Nela, informaremos ao Shiro, qual é a página de login do nosso sistema.

```
[main]
    authc.loginUrl=/faces/login.xhtml
```

Precisamos configurar quais usuários terão acesso ao nosso sistema. Faremos isso através da seção **users**, na qual definiremos um par *chave/valor* que será equivalente ao *Login/Senha*.

```
[users]
    leonardo=1234
```

Neste caso, cadastramos um usuário com o login **Leonardo** e com a senha **1234**.

Por último, solicitaremos ao *Shiro* o bloqueio da página `sorteio.xhtml` caso o usuário não tenha feito login.

```
[urls]
    /faces/sorteio.xhtml=authc
```

Por fim, nosso arquivo `shiro.ini` deve ter ficado desta forma:

```
[main]
    authc.loginUrl=/faces/login.xhtml
[users]
    leonardo=1234
[urls]
    /faces/sorteio.xhtml=authc
```

Precisamos agora autenticar o usuário quando ele fizer *login*. Voltemos ao método `login()` do nosso `ParticipanteBean`. Como queremos uma autenticação de usuário e senha, utilizaremos a classe do Shiro `UsernamePasswordToken`.

```
public void login() {
       UsernamePasswordToken token = new UsernamePasswordToken(participante.getEmail(), partici
}
```

Trabalhando com Subject

O Apache Shiro possui um mecanismo de controle de sessão independente. Por isso precisaremos um objeto que nos permita realizar operações nessa *Session*, como adicionar usuários (login) e removê-los (logout). Este objeto será uma instância de `Subject`: Uma classe do Shiro que nos permite realizar operações em sua sessão.

```
public void login() {
       UsernamePasswordToken token = new UsernamePasswordToken(participante.getEmail(), partici
       Subject user = ...
}
```

Para conseguirmos uma instância de `Subject` utilizaremos a classe útil `SecurityUtils`:

```
public void login() {
       UsernamePasswordToken token = new UsernamePasswordToken(participante.getEmail(), partici
       Subject user = SecurityUtils.getSubject();
}
```

Com uma instância de `Subject` em mãos, chamaremos o método `login` passando o nosso token de autenticação.

```
public void login() {
        UsernamePasswordToken token = new UsernamePasswordToken(participante.getEmail(), partici
        Subject user = SecurityUtils.getSubject();
        user.login(token);
}
```

Após realizarmos o login, iremos redirecionar o cliente para a página de sorteios. Para tal é preciso alterar o método login() e usar o string `sorteio?faces-redirect=true` no retorno do método.

```
public String login() {
        UsernamePasswordToken token = new UsernamePasswordToken(participante.getEmail(), partici
        Subject user = SecurityUtils.getSubject();
        user.login(token);

        return "sorteio?faces-redirect=true";
}
```

Vamos testar nosso login: ao entrarmos com os dados e realizarmos o *login*, somos redirecionados para a página de *Sorteio*. Agora iremos testar nosso login, inserindo dados inválidos. Repare que recebemos uma `UnknownAccountException` que é filha de `AuthenticationException` .

Vamos tratar essa *exception* fazendo um *try/catch* no método `login` .

```
public String login() {
    try {
        UsernamePasswordToken token = new UsernamePasswordToken(participante.getEmail(), partic
        Subject user = SecurityUtils.getSubject();
        user.login(token);

        return "sorteio?faces-redirect=true";
    } catch(AuthenticationException e) {
            . . .
    }
    return null;
}
```

Vamos exibir uma mensagem ao usuário dizendo que ocorreu um erro de autenticação. Para isso, exibiremos um `FacesMessage` .

```
public String login() {
    try {
        UsernamePasswordToken token = new UsernamePasswordToken(participante.getEmail(), partic
        Subject user = SecurityUtils.getSubject();
        user.login(token);

        return "sorteio?faces-redirect=true";
    } catch(AuthenticationException e) {
```

```
                FacesContext.getCurrentInstance().addMessage(null, new FacesMessage("Email ou se
        }
    return null;
}
```

Ao testarmos nosso login, podemos ver que a mensagem ainda não aparece. Isso aconteceu porque ainda não definimos onde será exibida essa mensagem. Portanto, devemos adicionar a tag `<p:messages />` no nosso `login.xhtml`

```
<p:messages />
<h:body>
    <h:form>
        <p:panelGrid columns="2">
            <p:outputLabel value="Email: "/>
            <p:inputText value="#{participanteBean.participante.email}" />

            <p:outputLabel value="Senha: " />
            <p:password value="#{participanteBean.participante.senha}" />

            <p:commandButton value="Logar" action="#{participanteBean.login}" ajax="false" />

        </p:panelGrid>
    </h:form>
</h:body>
```

CDI e producers

Para melhorar nosso código, vamos pedir ao CDI um *FacesMessage* e um *Subject* do Shiro. Mas para isso, devemos ensiná-lo a criar esses objetos. Vamos criar uma classe chamada de `ProducerUtils` :

```
public class ProducerUtils {
}
```

Nesta classe, criaremos um método que irá instanciar e devolver um *FacesContext*.

```
public class ProducerUtils {
   public FacesContext facesContextProducer() {
         return FacesContext.getCurrentInstance();
   }
}
```

Agora devemos dizer ao CDI que ao precisar injetar um objeto do tipo `FacesContext` , ele poderá buscá-lo através deste método. Para isso, vamos anotar o método com `@Produces` .

```
public class ProducerUtils {
   @Produces
   public FacesContext facesContextProducer() {
         return FacesContext.getCurrentInstance();
   }
}
```

Iremos fazer a mesma coisa para o `Subject` do Shiro.

```
public class ProducerUtils {
   @Produces
   public FacesContext facesContextProducer() {
         return FacesContext.getCurrentInstance();
   }
   @Produces
   public Subject subjectProducer() {
         return SecurityUtils.getSubject();
   }
}
```

Agora que o CDI já sabe criar esses objetos, podemos pedir para que ele os injete para nós. Para isso, na classe `ParticipanteBean` vamos criar os atributos:

```
private Subject user;
private FacesContext ctx;
```

E usar a anotação `@Inject` .

```
@Inject
private Subject user;
@Inject
private FacesContext ctx;
```

Vamos testar nossa aplicação e verificar que tudo continua funcionando corretamente.